

Obfuscating Keystroke Time Intervals to Avoid Identification and Impersonation

John V. Monaco
U.S. Army Research Laboratory
Aberdeen, MD
john.v.monaco2.ctr@mail.mil

Charles C. Tappert
Pace University
Pleasantville, NY
ctappert@pace.edu

Abstract

There are numerous opportunities for adversaries to observe user behavior remotely on the web. Additionally, keystroke biometric algorithms have advanced to the point where user identification and soft biometric trait recognition rates are commercially viable. This presents a privacy concern because masking spatial information, such as IP address, is not sufficient as users become more identifiable by their behavior. In this work, the well-known Chaum mix is generalized to a scenario in which users are separated by both space and time with the goal of preventing an observing adversary from identifying or impersonating the user. The criteria of a behavior obfuscation strategy are defined and two strategies are introduced for obfuscating typing behavior. Experimental results are obtained using publicly available keystroke data for three different types of input, including short fixed-text, long fixed-text, and long free-text. Identification accuracy is reduced by 20% with a 25 ms random keystroke delay not noticeable to the user.

1. Introduction

The issue of privacy in behavior monitoring has recently started to gain attention. It is not clear how many organizations routinely track user behavior and to what extent this information is being used. On the Internet, the situation is further complicated by the prevalence of third-party content which allows identifying attributes and user behavior to be seen by a number of websites as a result of visiting only a single page [15]. Behavior tracking is motivated by targeted advertising, analytics, and law enforcement. The laws and regulations currently surrounding behavior tracking are relatively lax, although a majority of users negatively view be-

havioral targeting. In 2012, a study by Pew Research Group found 68% of respondents to be “not okay” with behavioral targeted advertising, and a 2010 USA Today/Gallup poll showed 67% of respondents felt that behavioral targeting should not be legal [3, 6]. Behavior tracking capabilities are extended when user keystrokes are added to the mix [4].

The simplicity and ubiquitous nature of keystroke behavior makes it attractive as a biometric modality for computer users. There are numerous commercial applications and government interest in the area of keystroke biometrics. To name a few, keystroke biometrics has recently been considered as a means of offering students verified certificates for completing a massively open online course (MOOC) [13]. Funding through DARPA’s Active Authentication program has helped the field advance considerably, creating partnerships between government, academia, and industry [7]. One of the most commercially successful keystroke biometric applications is delivered by BehavioSec, a Swedish company that utilizes typing behavior, among other factors, to verify the legitimacy of online transactions. In 2015, BehavioSec verified over 1.5B transactions from 15M users across 20 different banks [9].

Behavioral patterns and identifying attributes may be unintentionally leaked through keystroke timing information, which can be remotely observed without a victim’s knowledge or consent. Masking spatial information, such as IP address through TOR [27], is futile when temporal information, such as keystroke timings, can be used for identification. The consequences of an adversary being able to observe a victim’s keystrokes are twofold. First, the victim may be identified by their typing behavior. Even without having previously observed the victim’s typing behavior, identifying attributes such as age, gender, handedness, and native language, can be resolved with reasonable accuracy [2, 8]. Second, the victim may be impersonated through a generative model of typing behavior, potentially enabling an adversary to gain access to a system that implements keystroke biometric access control.

This paper discusses the implications of typing behavior that is observable by a third party over the web and pro-

This research was supported in part by an appointment to the Postgraduate Research Participation Program at the U.S. Army Research Laboratory administered by the Oak Ridge Institute for Science and Education through an inter agency agreement between the U.S. Department of Energy and USARL.

poses several mechanisms aimed at preserving anonymity. The goals, constraints, and theoretical limits of keystroke behavior obfuscation are explored. Section 2 provides further motivation and rationale for obfuscating typing behavior and Section 3 reviews some background material and related work. Two obfuscation strategies are introduced in Section 4 followed by an empirical evaluation using three types of keystroke input from publicly available databases in Section 5. Finally, Section 6 concludes the paper.

2. Motivation

Humans generate events over a wide range of time scales separated by orders of magnitude, as exemplified by Newell’s time scale of human action [19]. At the lowest level are biological events, e.g., the firing of individual neurons on the order of microseconds, and at the highest level are group and social dynamics which emerge as a result of higher-frequency events. There is generally less privacy as one moves up (down in frequency) in Newell’s time scale. For example, high frequency brain and heart activity, measured through electroencephalogram (EEG) and electrocardiogram (ECG), require specialized sensors and the presence of the user. On the other hand, lower frequency events such as email, financial transactions, and mobile device interactions, can typically be observed by a remote third party. Keystrokes fall in the cognitive band, which lies between the biological and social bands and where events occur on the order of milliseconds to seconds. Since keystrokes are ubiquitous with modern computers and often contain sensitive information, they are the subject of some privacy issues.

The standard desktop computer measures key-press and key-release events with a clock precision of about 16 ms and resolution¹ of at least 1 ms [11]. Keystroke events are typically recorded by a keylogger in either a system-wide context via a hook registered in the kernel or in a sandbox environment, such as within a single page of a web browser. In some cases, simply disabling the logging software on the client is sufficient to ensure that a third party cannot observe the victim’s keystrokes. An exception to this rule comes in the form of interactive applications, which generate network traffic immediately after a key is pressed. Even if third party logging software is disabled on the client, e.g., only trusted Javascript sources are enabled, network timing information can be used to build a profile of typing behavior since keystrokes result in the immediate transmission of packets over a network. Key-press times can be obtained remotely without installing a keylogger on the victim’s computer by observing the network traffic generated from an interactive application, such as SSH in interactive mode [24] and Google Suggestions service [26]. A timing attack on

SSH network traffic timestamps collected during password entry can provide about 1 bit of information in cracking a password. [24] verified that the key-press latencies can be reliably determined from the packet inter-arrival times from interactive SSH traffic, as the time between the actual press of a key and packet creation by the kernel is negligible.

2.1. Identification

Identification is performed by enrolling a user’s keystroke samples into a database during a trusted session. The samples make up the user’s keystroke template. For fixed input, such as a password, this may take the form of various key hold durations (the time from key press to key release) and key press/key release latencies (the time between successive key presses or key release to key press), such as in [12]. For free-text input, a set of descriptive statistics on the various time intervals may be defined, such as in [25]. Note that in this work, only timing information is considered. Some works that deal with free-text input also consider linguistic features in an attempt to capture the author’s stylometry, or writing style. This work does not attempt to mask information that may be leaked through linguistic analysis, although there have been other attempts in successfully doing so [10].

The identification of soft biometric traits without a user’s prior enrollment is also a real possibility. In this scenario, a user’s typing behavior may reveal soft attributes such as age, gender, handedness, and native language. Such an attack is feasible with several publicly available databases that contain soft biometric labels and the relative ease in collecting a new database with the desired labels through, e.g., Amazon’s Mechanical Turk. In [8], age, gender, and handedness are classified with between 80% and 100% accuracy using both fixed text and free text input. This database is made publicly available. In [2], handedness, gender, and native language are classified with accuracies that significantly deviate from chance.

2.2. Impersonation

Impersonation is performed by mimicking keystroke behavior with the intent of being recognized as the victim. This represents a non-zero-effort attack on behalf of an impostor. With access to the victim’s computer, a spoofing attack such as in [22] can be utilized. Observing a victim’s keystrokes directly allows for typing behavior to be easily replicated. Without any knowledge of the victim’s typing pattern, an attack such as [23] can be used. The latter work used a template enumeration technique, in which an independent keystroke database was used to reduce the search space of typing behavior. This works well for shorter strings and assumes that the verification system allows multiple attempts.

[18] provides empirical evidence for a two-state gener-

¹Clock resolution is the degree to which a measurement can be made and clock precision is the degree to which a measurement can be repeated.

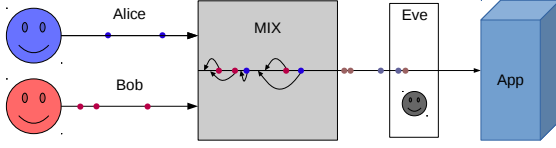


Figure 1. Chaum mix. Eve is able to observe a single arrival process and must discern which user owns each event.

ative model of typing behavior in which the user can be in either a passive or active state. Given key-press latencies with missing key names, the model is then used to predict the key-press latencies of a user by exploiting the linear relationship between inter-key distance and key-press latency. The proposed generative model uses this partial information to perform a key-press-only sample-level attack on a victim’s keystroke dynamics template. Results show that some users are more susceptible to this type of attack than others. For about 10% of users, the spoofed samples obtain classifier output scores of at least 50% of those obtained by authentic samples, and with at least 50 observed keystrokes the chance of successful verification over a zero-effort attack doubles on average.

3. Background

There are primarily two goals in obfuscating keystroke behavior. The first is to limit an adversary’s capability to identify a user. Under this goal, the user wishes to remain anonymous within a pool of U users. Anonymity is achieved when the probability of correctly identifying the user is no greater than $\frac{1}{U}$. With cooperation from all the users in the pool, it is relatively easy to obtain perfect anonymity for every user. Each user needs only to behave in some predefined way agreed upon by the pool. If the behavior of every user is exactly the same, the best strategy for identification is simply a random guess out of the U users.

The second goal is to limit an adversary’s capability to predict user behavior, which is necessary for impersonation. This goal is quite different, in that the user wishes for the time and duration of a future keystroke to be unpredictable after having generated N keystrokes. Perfect unpredictability is achieved when the expected symmetric mean absolute percentage error (SMAPE) of the predictions go to 1 [21]. The SMAPE can go to 1 when either the actual time interval between events, τ_{N+1} , or predicted time interval between events, $\hat{\tau}_{N+1}$, is infinite. The actual time interval is infinite when the user plans on waiting indefinitely before generating another keystroke. The predicted time interval is infinite if the adversary’s model predicts an infinite estimated delay until the next keystroke. Neither of these situations are practical and thus perfect unpredictability can generally not be achieved.

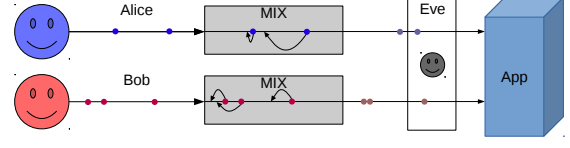


Figure 2. User mix. Eve is able to observe multiple arrival processes and must discern which user owns each process.

Both goals in obfuscating keystroke behavior call for different strategies. Consider two extreme examples. In the first, every user decides to generate keystrokes with exactly the same frequency. That is, the time between events τ is the same for every user. Under this strategy, perfect anonymity is achieved. However, each user’s actions can also be predicted exactly and the masking strategy fails to address the predictability criterion. Consider a different strategy, in which each user generates keystrokes according to a Poisson process. Let the users have rates $\lambda_1 \ll \dots \ll \lambda_U$. On a per-user basis, predictive accuracy is relatively low, as the time intervals are independent and identically distributed, following an exponential distribution. However, users can be easily identified over time by their expected time interval, or event frequency.

A more effective strategy would be to let each user generate events according to a Poisson process with rate λ . With cooperation from every user, perfect anonymity is achieved as the Poisson process is memoryless and the resulting distributions of time intervals from each user will appear to be the same. Unfortunately, this strategy is generally not practical since it requires the cooperation from every user. Some users who naturally type at a rate much lower than λ would be forced to increase their speed. Conversely, users who naturally type at a rate much higher than λ would be forced to slow down.

Chaum mixes can be used to provide anonymity to users behind a router, in which cooperation is assumed [5]. This scenario is shown in Figure 1, where users Alice and Bob are behind a router and wish to transmit a series of packets. The *mix* reorders the packets by introducing a random delay to each packet. An eavesdropper, Eve, observes the sequence of reordered packets before they reach the application, however cannot discern whether each packet came from Alice or Bob. The identity and temporal behavior of Alice and Bob are protected by the mix.

The anonymity of the mixing strategy is given by the expected entropy of the observed packets,

$$\mathcal{A} = \lim_{N \rightarrow \infty} \frac{1}{N} \mathbb{E}[H(U_1, \dots, U_N)] \quad (1)$$

where U_n is the identity of the n^{th} observed packet and H is the entropy function. Without any constraints, the

optimal strategy that maximizes anonymity can be shown to require an infinite delay [16]. This is due to the mix waiting for all packets from both Alice and Bob and then randomly selecting one of the $\binom{2N}{N}$ permutations for the packet reordering. This strategy is not practical for a number of reasons, one being that it would require an infinite buffer size. With a finite buffer size, the upper bound on the maximum-achievable anonymity decreases. Despite this, a reasonable level of anonymity can be achieved with practical constraints [28].

The scenario considered in this work is a generalization of the Chaum mix, shown in Figure 2. Since the Chaum mix requires the cooperation of all the users, i.e., every user must be on the same network, it is a global obfuscation strategy, or *global mix*. If the assumption of cooperation is relaxed, then an obfuscation strategy without cooperation must be developed. This type of strategy is provided by a *user mix*. A user mix is appropriate when there is no cooperation between users, or the events from different users are separated by both space and time².

Compared to the Chaum mix that operates globally on events from every user, the user mix operates exclusively on events from only one user. The mix for each user is atomic, in the sense that it operates independently of every other user. Different users may employ different obfuscation strategies or no strategy at all. The mix separates the *generating process*, the sequence of events from the perspective of the user, from the *arrival process*, the sequence of events as seen by the application. The goal of the mix is to provide anonymity and unpredictability at the arrival process.

This scenario is depicted in Figure 2, where Alice and Bob both generate events and wish to remain anonymous and unpredictable. An eavesdropper, Eve, is able to observe the sequence of events from each user after passing through the mix, and she knows that there are two users in the system. From the perspective of Eve, it is not clear which sequence belongs to each user, as she can only observe the arrival processes. The technical conditions of the user mix are summarized as follows.

1. Alice and Bob have zero knowledge of each other's behavior and don't cooperate.
2. Eve can see multiple arrival processes, but cannot discern which process belongs to each user
3. Each mix has a source of randomness unknown to Eve.
4. Event order is preserved by the mix.

4. Obfuscation strategies

The user mix delays events by temporarily storing them in a buffer before releasing them to the application. With-

²For example, consider keystrokes recorded by a web application. The keystrokes from two different sessions could have been recorded days apart and come from entirely different network locations.

| | |
|----------------------|--|
| Finite | The expected delay between the user and the arrival process should not grow unbounded. |
| Anonymous | The mix should make it difficult to identify the user. |
| Unpredictable | The mix should make it difficult to predict future behavior. |

Table 1. Desirable properties of a user mix.

out access to future events, and under the constraint that events cannot be permuted, the only operation that can be performed is to delay an event. An event consists of a key press or key release. The constraint that events cannot be permuted ensures that the characters appear at the arrival process in the order and form (e.g., case or special symbol) they were generated.

There are a few caveats in obfuscating temporal behavior. Since the only action is to delay, a lag is introduced between the generation of an event by the user and the observation of the event in the application. Thus, it may not be possible to use a mix in some real-time systems. For many human-computer interaction applications, a small lag is acceptable and would not be noticed by the user. As the lag increases, the movement time and error rate on behalf of the user also increase [14].

A user mix should possess several properties, summarized in Table 1. The expected lag between the generated events and observed events should be finite. The size of the event buffer ultimately determines how large the lag can grow to. If the tolerated lag is unbounded, then the number of events that need to be stored will eventually exceed the size of the buffer. A user mix should also protect the user from being identified out of a population of users. This condition is more difficult to satisfy, as it requires at least some global knowledge of other users in the system. A mix that provides anonymity will emulate the temporal behavior of the “typical” user. Finally, a mix should make it difficult to reproduce and predict the temporal behavior of a user.

Analogous to the tradeoff between type I and type II errors on the ROC curve, there is a direct tradeoff between time lag and the obfuscation capability of the mix. As the lag decreases, the dependence between the arrival process and the generating process increase. As the lag increases, the potential to reduce dependence increases. It is desirable to have a mix that maximally reduces the dependence between the generating process and the arrival process for a given lag.

Let t_n^o and τ_n^o be the true time and time interval of the n^{th} event generated by the user, and t_n and τ_n be the observed time and time interval of the n^{th} event after passing through the mix. The sequence of t_n , or alternatively τ_n , constitute the arrival process. The lag, or delay, between the generated event and observed event is given by δ_n . Since event order must be preserved by the mix, it is necessary that $t_{n-1} +$

| Symbol | Description |
|----------------|--|
| t_n | time of events at the arrival process |
| t_n° | time of events at the generating process |
| τ_n | time interval between events at the arrival process |
| τ_n° | time interval between events at the generating process |
| δ_n | time lag between the generating and arrival processes |

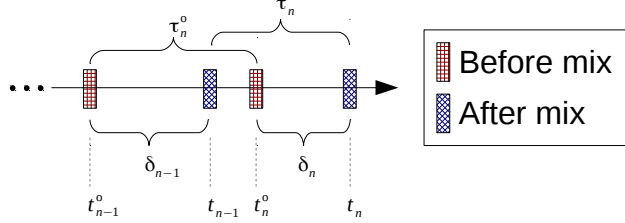


Figure 3. Summary of user mix operation and variables.

$\delta_{n-1} < t_n + \delta_n$. Figure 3 summarize the variables of the user mix.

Predictability can be measured by the dependence between τ° and τ . Anonymity is given by the dependence between τ^A and τ^B from two random instantiations of the mix, A and B . Dependence is measured by the mutual information between two continuous random variables, given by

$$I(X, Y) = \int_Y \int_X f(x, y) \log \left[\frac{f(x, y)}{f(x)f(y)} \right] dx dy \quad (2)$$

where $f(x, y)$ is the joint probability density function of x and y .

A mix with that provides perfect unpredictability has $I(\tau^\circ, \tau) = 0$. Perfect anonymity is achieved when $I(\tau^A, \tau^B) = 0$. Both can be achieved with an unlimited buffer size and delay. To demonstrate this, consider a mix that samples τ from a uniform distribution, i.e., $\tau \sim \mathcal{U}(0, k)$. Let τ_n be the time interval of the arrival process. If $\tau_{n+1}^\circ < \tau_n$, then the lag between the user and the arrival process must increase by at least $\tau_n - \tau_{n+1}^\circ$ and as $n \rightarrow \infty$, the lag increases unbounded. In this case, $I(\tau^\circ, \tau) = 0$ since $f(\tau^\circ, \tau) = f(\tau^\circ)f(\tau)$, i.e., the observed time intervals are independent from the actual time intervals. If $\tau_{n+1}^\circ > \tau_n$, then the time interval on the arrival process is no longer independent from the actual time intervals since $\tau_{n+1} > \tau_{n+1}^\circ - \tau_n$. The next sample will have $\tau_{n+1} \sim \mathcal{U}(\tau_{n+1}^\circ - \tau_n, \tau_{n+1}^\circ - \tau_n + 1)$, or alternatively $P(\tau_{n+1} = s) = \frac{1}{k}$ where $\tau - \tau_n < s < \tau_{n+1}^\circ - \tau_n + k$. Consequently, the dependence guarantees that $f(\tau^\circ, \tau) > f(\tau^\circ)f(\tau)$ and $I(\tau^\circ, \tau) > 0$. Therefore, in practice it is necessary to introduce a lag constraint in minimizing the dependence between τ° and τ .

There are primarily two ways a time interval mix can add noise to the time intervals of the arrival process. Let δ be the lag between τ° and τ . The first is by generating a random delay between the actual events that occur at time

Algorithm 1 Delay mix.

1. **Initialize:** Let $\tau_0^\circ = \infty$ and $t_0^\circ = 0$
2. **Generate:** Let $l_n = \max(\delta_{n-1} - \tau_n^\circ, 0)$ be the lower bound of the n^{th} delay and $\delta_n \sim \mathcal{U}(l, \Delta)$ be a random delay. The event time on the arrival process is given by $t_n = t_n^\circ + \delta_n$

t° resulting in the time $t^\circ + \delta$ at the arrival process. This type of mix is a *delay mix*. The second way is to generate τ directly, referred to as an *interval mix*.

4.1. Delay mix

The delay mix introduces noise to the time intervals of the arrival process by randomly delaying each event. For the n^{th} event at time t_n° , the delay mix generates a random delay δ_n to produce the event time at the arrival process, $t_n = t_n^\circ + \delta_n$. Time intervals of the arrival process are given by $\tau_n = (t_n^\circ + \delta_n) - (t_{n-1}^\circ + \delta_{n-1})$. A uniform distribution can be used to generate delays. Let $\delta_n \sim \mathcal{U}(\max(\delta_{n-1} - \tau_n^\circ, 0), \Delta)$, where Δ is the upper bound on the delay. The lower bound of $\max(\delta_{n-1} - \tau_n^\circ, 0)$ is necessary to ensure the event order is preserved. Without this constraint, events will become permuted if $\tau_n^\circ + \delta_n < \delta_{n-1}$. The process is summarized in Algorithm 1.

The expected delay is bounded above by Δ . To see this, consider the two scenarios: $\tau_n^\circ < \delta_{n-1}$ and $\tau_n^\circ > \delta_{n-1}$. If $\tau_n^\circ < \delta_{n-1}$, then the expected lag is simply $E[\delta_n] = \frac{\Delta}{2}$ since the lower bound will always be 0. If $\tau_n^\circ > \delta_{n-1}$, then $E[\delta_n] = \frac{\Delta + \delta_{n-1} - \tau_n^\circ}{2}$ which is bounded above by Δ . Given $\tau_0 = \infty$, it is true that $0 < \delta_1 < \Delta$. By induction, it is also true that $\delta_{n-1} < \Delta$ since $\tau_n^\circ > 0$.

4.1.1 Example

As an example, consider events occurring at times $t^\circ = [0, 5, 7, 11, 14]$ passing through a delay mix with $\Delta = 7$. The actual time intervals are $\tau^\circ = [\infty, 5, 2, 4, 3]$. The trace of each variable is shown in Table 2.

| Event | t° | τ° | l | δ | t | τ |
|-------|-----------|--------------|-----|----------|-----|----------|
| 0 | 0 | ∞ | 0 | 3 | 3 | ∞ |
| 1 | 5 | 5 | 0 | 6 | 11 | 8 |
| 2 | 7 | 2 | 4 | 5 | 12 | 1 |
| 3 | 11 | 4 | 1 | 5 | 16 | 4 |
| 4 | 14 | 3 | 6 | 6 | 20 | 4 |

Table 2. Delay mix example where t° and τ° are the actual time and time interval of the n^{th} event, l is the lower bound of the delay, δ is the random delay, and t and τ are the time and time interval of the arrival process.

Algorithm 2 Interval mix.

1. **Initialize:** Let $t_0^\circ = t_0 = 0$, $\tau_0^\circ = \tau_0 = \infty$, and $u_1 > 0$.
 2. **Generate:** Let $\dot{\tau}_n \sim \mathcal{U}(0, u_n)$ be the desired time interval and $\dot{t}_n = t_{n-1} + \dot{\tau}_n$ be the desired time of the arrival process. The arrival process time is given by $t_n = \max(\dot{t}_n, t_n^\circ)$.
 3. **Update:** Let $u_{n+1} = \max[u_n + b(t_n^\circ - \dot{t}_n), \epsilon]$
-

4.2. Interval mix

Instead of generating delays, the time intervals of the arrival process can be modeled explicitly. The goal of the mix is to minimize the dependence between τ° and τ , and this strategy will give the mix greater control over the resulting time intervals. While the delay mix generates δ_n at each time step to produce arrival process time $t_n = t_n^\circ + \delta_n$, the interval mix generates τ_n to produce $t_n = t_{n-1} + \tau_n$. With an infinite delay, this allows the τ_n to be generated independently of τ_n° . With a finite delay, the τ_n are constrained by the event rate of the user.

At each time step t_n , let the desired time interval be $\dot{\tau}_n \sim \mathcal{U}(0, u_{n-1})$ where u_{n-1} is an upper bound parameter. If $t_{n-1} + \dot{\tau}_n > t_n^\circ$, then the n^{th} event is delayed by $\delta_n = t_{n-1} + \dot{\tau}_n - t_n^\circ$ to get the arrival process time $t_n = t_{n-1} + \dot{\tau}_n$. If $t_{n-1} + \dot{\tau}_n \leq t_n^\circ$, then $\delta_n = 0$ and the event is released immediately. The upper bound u is updated as $u_n = \max\{u_{n-1} + b[t_n - (t_{n-1} + \dot{\tau}_n)], \epsilon\}$, where b is a parameter that controls the rate at which u_n can change. This update moves the expected time interval in the direction of the rate of the user. This process is summarized in Algorithm 2.

4.2.1 Example

An example is used to demonstrate the interval mix with parameter $b = 1$. Similar as before, the actual event times are given by $t^\circ = [0, 5, 7, 11, 14]$ with time intervals $\tau^\circ = [\infty, 5, 2, 4, 3]$. At the first event, there is no delay between the actual event time and the arrival process, therefore $t_0 = t_0^\circ = 0$. The parameter updates and time intervals of the arrival process are shown in Table 3. For simplicity, values are sampled from a discrete uniform distribution. The starting value of u is chosen to be 7 to demonstrate the adaptability of the mix to the user’s behavior in several iterations. The starting value of u can be any positive value and will quickly converge to an appropriate range.

| Event | t° | τ° | u | \dot{t} | $\dot{\tau}$ | δ | t | τ |
|-------|-----------|--------------|-----|-----------|--------------|----------|-----|----------|
| 0 | 0 | ∞ | - | - | - | 0 | 0 | ∞ |
| 1 | 5 | 5 | 7 | 3 | 3 | 0 | 5 | 5 |
| 2 | 7 | 2 | 9 | 11 | 6 | 4 | 11 | 6 |
| 3 | 11 | 4 | 5 | 15 | 4 | 4 | 15 | 4 |
| 4 | 14 | 3 | 1 | 16 | 1 | 2 | 16 | 1 |

Table 3. Interval mix example where t° is the actual event time, τ° is the actual time interval, u is the interval distribution parameter, \dot{t} is the desired event time, t is the arrival process event time, and τ is the arrival process time interval.

5. Case study

The two time interval obfuscation strategies introduced in Section 4 are empirically evaluated using publicly available keystroke datasets with previously published results. Three different types of keystroke input are considered: short fixed-text (e.g., password or PIN), long fixed-text (e.g., copying or transcribing several sentences), and long free-text (e.g., response to an open-ended question).

Each dataset contains labeled attributes for identity, age, handedness, and gender. The short fixed-text comes from [8], the long fixed-text from [1], and the long free-text from [17]. The short fixed-text dataset contains 110 users with 50 samples per user. Each sample contains one entry in which the user typed a short passphrase ranging from 17 to 24 characters as described in [8]. The long fixed-text dataset is a combination of the fixed text from [1] and [17], during which users copied short sentences and fables. The combined dataset contains 101 users with 10 samples per user and 123 ± 38 characters per sample. The free text dataset is also from [17] and contains 127 users with 10 samples per user. Users were required to respond to open-ended and essay style questions. The long responses were sliced to create samples of length 123 ± 38 to match the distribution of the long fixed-text dataset.

Classification of each target variable is performed by a random forest classifier with features and fallback hierarchy as described in [17]. The RandomForest classifier in the sklearn Python package is utilized with 200 estimators and the same features for all three types of keystroke input. Classification accuracy (ACC) is obtained by a stratified 10 fold cross validation for the identity target. For the age, handedness, and gender targets, a 110, 101, and 127 cross-fold validation is used for the short fixed-text, long fixed-text, and long free-text, respectively. In each fold, a single user’s samples are used as the testing set and the remainder of the population as the training set. This emulates an open system, in which soft attributes for the target user are classified without having previously observed that user’s keystrokes. The sizes of each class in the training sets are not altered. Thus, the baseline prediction rates coincide with the proportion of the largest class, which are 0.51 (age ≥ 30), 0.68 (male), 0.88 (right

| | Δ | δ | Id. | Age | Gen. | Han. | PP | DU |
|------------------|----------|----------|------|------|------|------|------|------|
| Short fixed-text | 0 | 0 | 0.55 | 0.69 | 0.75 | 0.70 | 0.20 | 0.14 |
| | 50 | 26 | 0.41 | 0.73 | 0.77 | 0.70 | 0.21 | 0.18 |
| | 100 | 54 | 0.27 | 0.69 | 0.77 | 0.70 | 0.22 | 0.26 |
| | 200 | 123 | 0.16 | 0.76 | 0.77 | 0.70 | 0.25 | 0.32 |
| | 500 | 391 | 0.12 | 0.72 | 0.77 | 0.70 | 0.27 | 0.35 |
| | 1000 | 872 | 0.11 | 0.73 | 0.77 | 0.70 | 0.30 | 0.38 |
| Long fixed-text | 0 | 0 | 0.86 | 0.64 | 0.58 | 0.80 | 0.31 | 0.16 |
| | 50 | 26 | 0.79 | 0.65 | 0.56 | 0.80 | 0.31 | 0.19 |
| | 100 | 55 | 0.67 | 0.66 | 0.58 | 0.80 | 0.32 | 0.26 |
| | 200 | 129 | 0.53 | 0.62 | 0.63 | 0.79 | 0.34 | 0.31 |
| | 500 | 404 | 0.46 | 0.68 | 0.68 | 0.80 | 0.34 | 0.34 |
| | 1000 | 891 | 0.44 | 0.66 | 0.67 | 0.79 | 0.34 | 0.35 |
| Long free-text | 0 | 0 | 0.71 | 0.72 | 0.78 | 0.70 | 0.30 | 0.15 |
| | 50 | 26 | 0.62 | 0.69 | 0.76 | 0.70 | 0.31 | 0.18 |
| | 100 | 54 | 0.51 | 0.75 | 0.77 | 0.70 | 0.32 | 0.24 |
| | 200 | 126 | 0.36 | 0.68 | 0.71 | 0.70 | 0.33 | 0.30 |
| | 500 | 394 | 0.30 | 0.69 | 0.72 | 0.70 | 0.33 | 0.33 |
| | 1000 | 876 | 0.28 | 0.68 | 0.74 | 0.70 | 0.34 | 0.35 |

Table 4. Delay mix experimental results. Id.=identity, Gen.=gender, Han.=handedness, PP=press-latency SMAPE, DU=duration SMAPE.

handed) for age, gender, and handedness targets for all three datasets combined. Predictions are made for both the press-latency and duration using the mean time interval up to the observed event, $\widehat{PP}_n = \langle PP_1^{n-1} \rangle$ and $\widehat{DU}_n = \langle DU_1^{n-1} \rangle$ where PP_n and DU_n are the press-latency and duration of the n^{th} keystroke, respectively. Source code for the experiments in this work is available at <https://github.com/vmonaco/keystroke-obfuscation>.

Using each dataset, classification accuracies for each target variable are obtained before and after applying each masking strategy. The mean lag δ between the generating process and arrival process is also calculated for each parameter choice. Table 4 contains experimental results using the delay mix for various values of Δ and Table 5 contains experimental results using the interval mix for various values of b . The classification accuracies of each mix as a function of the mean lag δ are summarized in Figure 4.

6. Conclusions

The results in this work suggest that it is possible to obfuscate keystroke behavior with a delay that is not noticeable to the user, however the time lag between the user and the application remains a practical constraint. With a 25 ms delay, identification accuracy is reduced by approximately 20% on average, and in most cases a 500 ms delay is needed to halve the identification accuracy. Soft biometric trait classification accuracies, which are initially near chance accuracy, are relatively unaffected by the obfuscated keystrokes. Results may differ in a scenario where, e.g., only the unlabeled testing data is obfuscated. Prediction errors of the duration and press-latency increase as expected using both types of obfuscation strategy.

An important point is that the proposed methods operate

| | b | δ | Id. | Age | Gen. | Han. | PP | DU |
|------------------|-----|----------|------|------|------|------|------|------|
| Short fixed-text | 0.0 | 0 | 0.55 | 0.69 | 0.75 | 0.70 | 0.20 | 0.14 |
| | 0.1 | 24 | 0.33 | 0.72 | 0.77 | 0.70 | 0.19 | 0.18 |
| | 0.5 | 50 | 0.21 | 0.74 | 0.76 | 0.70 | 0.24 | 0.27 |
| | 1.0 | 70 | 0.17 | 0.75 | 0.78 | 0.70 | 0.29 | 0.36 |
| | 1.5 | 90 | 0.15 | 0.72 | 0.77 | 0.70 | 0.33 | 0.43 |
| | 2.0 | 117 | 0.13 | 0.74 | 0.78 | 0.70 | 0.37 | 0.49 |
| Long fixed-text | 0.0 | 0.00 | 0.86 | 0.64 | 0.58 | 0.80 | 0.31 | 0.16 |
| | 0.1 | 471 | 0.57 | 0.63 | 0.64 | 0.80 | 0.32 | 0.26 |
| | 0.5 | 1566 | 0.50 | 0.65 | 0.66 | 0.79 | 0.40 | 0.37 |
| | 1.0 | 2544 | 0.42 | 0.59 | 0.65 | 0.79 | 0.46 | 0.46 |
| | 1.5 | 3570 | 0.35 | 0.64 | 0.66 | 0.80 | 0.50 | 0.53 |
| | 2.0 | 4218 | 0.32 | 0.65 | 0.59 | 0.80 | 0.54 | 0.58 |
| Long free-text | 0.0 | 0 | 0.71 | 0.72 | 0.78 | 0.70 | 0.30 | 0.15 |
| | 0.1 | 385 | 0.37 | 0.64 | 0.75 | 0.70 | 0.32 | 0.26 |
| | 0.5 | 1093 | 0.30 | 0.63 | 0.75 | 0.70 | 0.40 | 0.39 |
| | 1.0 | 1840 | 0.27 | 0.68 | 0.74 | 0.70 | 0.47 | 0.49 |
| | 1.5 | 2321 | 0.23 | 0.65 | 0.68 | 0.70 | 0.51 | 0.56 |
| | 2.0 | 3783 | 0.18 | 0.64 | 0.77 | 0.70 | 0.56 | 0.61 |

Table 5. Interval mix experimental results.

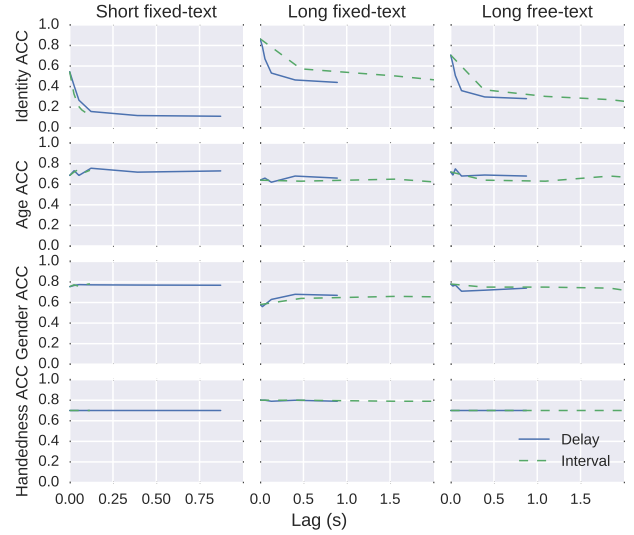


Figure 4. Time lag vs classification accuracy for each type of keystroke input and target variable.

at the keystroke event level and not at the network packet level. Only in certain situations do keystroke events correspond to network packets, such as interactive web applications. Another realistic scenario is one in which Eve resides on a workstation, and Alice and Bob are users typing on a peripheral USB keyboard. The proposed obfuscation strategies could be implemented in hardware, e.g., as a device that sits between the keyboard and the computer. This device would ensure a given level of anonymity by randomly delaying the USB events before they reach the workstation. The hardware implementation may prove more secure than the software implementation, and it avoids interference with the application (such as disabling Javascript on a web page and subsequently losing other functionality).

Future work should investigate additional obfuscation

strategies and scenarios, such as training on normal data and testing obfuscated data. Additionally, a major limitation of this work is that the proposed obfuscation strategies ignore the possibility that only a single user utilizes a mix and can be easily detected. The susceptibility of additional features and classifiers should also be investigated with the proposed obfuscation strategies. Temporal behavior obfuscation is a relatively unexplored area. The described strategies can be applied to network traffic to hinder device fingerprinting and network traffic classification based on packet inter arrival times, such as those described in [20]. It can also be applied to lower-frequency events, such as financial transactions, although it is not clear if there would be any benefit in doing so.

References

- [1] L. Bello, M. Bertacchini, C. Benitez, J. C. Pizzoni, and M. Cipriano. Collection and publication of a fixed text keystroke dynamics dataset. In *XVI Congreso Argentino de Ciencias de la Computación*, 2010.
- [2] D. G. Brizan, A. Goodkind, P. Koch, K. Balagani, V. V. Phoha, and A. Rosenberg. Utilizing linguistically-enhanced keystroke dynamics to predict typist cognition and demographics. *International Journal of Human-Computer Studies*, 2015.
- [3] P. R. Center. Search engine use 2012, 2012. Accessed August 2015.
- [4] P. Chairunnanda, N. Pham, and U. Hengartner. Privacy: Gone with the typing! identifying web users by their typing patterns. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 974–980. IEEE, 2011.
- [5] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
- [6] Gallup. Usa today/gallup poll, 2010. Accessed August 2015.
- [7] R. P. Guidorizzi. Security: Active authentication. *IT Professional*, (4):4–7, 2013.
- [8] S. Z. S. Idrus, E. Cherrier, C. Rosenberger, and P. Bours. Soft biometrics for keystroke dynamics: Profiling individuals while typing passwords. *Computers & Security*, 45:147–155, 2014.
- [9] B. Inc. Behaviorsec in retail banking. Accessed January 2016.
- [10] G. Kacmarcik and M. Gamon. Obfuscating document stylometry to preserve author anonymity. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 444–451. Association for Computational Linguistics, 2006.
- [11] K. Killourhy and R. Maxion. The effect of clock resolution on keystroke dynamics. In *Recent Advances in Intrusion Detection*, pages 331–350. Springer, 2008.
- [12] K. S. Killourhy and R. A. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *Dependable Systems & Networks, 2009. DSN’09. IEEE/IFIP International Conference on*, pages 125–134. IEEE, 2009.
- [13] A. Maas, C. Heather, C. T. Do, R. Brandman, D. Koller, and A. Ng. Offering verified credentials in massive open online courses: Moocs and technology to advance learning and learning research (ubiquity symposium). *Ubiquity*, 2014(May):2, 2014.
- [14] I. S. MacKenzie and C. Ware. Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT’93 and CHI’93 conference on Human factors in computing systems*, pages 488–493. ACM, 1993.
- [15] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 413–427. IEEE, 2012.
- [16] A. Mishra and P. Venkitasubramaniam. The anonymity of an almost fair chaum mix. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 693–700. IEEE, 2011.
- [17] J. V. Monaco, N. Bakelman, S.-H. Cha, and C. C. Tappert. Recent advances in the development of a long-text-input keystroke biometric authentication system for arbitrary text input. In *European Intelligence and Security Informatics Conference (EISIC)*, pages 60–66. IEEE, 2013.
- [18] J. V. Monaco, C. C. Tappert, and M. L. Ali. Spoofing keypress latencies with a generative keystroke dynamics model. In *Biometrics: Theory, Applications and Systems (BTAS)*. IEEE, 2015.
- [19] A. Newell. *Unified theories of cognition*. Harvard University Press, 1994.
- [20] S. V. Radhakrishnan, A. S. Uluagac, and R. Beyah. Gtid: A technique for physical device and device type fingerprinting. 2014.
- [21] V. Raghavan, A. Galstyan, A. G. Tartakovsky, et al. Hidden markov models for the activity profile of terrorist groups. *The Annals of Applied Statistics*, 7(4):2402–2430, 2013.
- [22] K. A. Rahman, K. S. Balagani, and V. V. Phoha. Snoop-forge-replay attacks on continuous verification with keystrokes. *IEEE Transactions on Information Forensics and Security*, 8(3):528–541, 2013.
- [23] A. Serwadda and V. V. Phoha. Examining a large keystroke biometrics dataset for statistical-attack openings. *ACM Transactions on Information and System Security (TISSEC)*, 16(2):8, 2013.
- [24] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001, 2001.
- [25] C. C. Tappert, M. Villani, and S.-H. Cha. Keystroke biometric identification and authentication on long-text input. *Behavioral biometrics for human identification: Intelligent applications*, pages 342–367, 2009.
- [26] C. M. Tey, P. Gupta, D. Gao, and Y. Zhang. Keystroke timing analysis of on-the-fly web apps. In *Applied Cryptography and Network Security*, pages 405–413. Springer, 2013.
- [27] Tor. The tor project: Anonymity online, 2015. Accessed August 2015.
- [28] P. Venkitasubramaniam and V. Anantharam. On the anonymity of chaum mixes. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 534–538. IEEE, 2008.